Database Design Project

Phase 2 – Relational Schema Design

Gary Steelman | [gxs112030@utdallas.edu](mailto:gxs112030@utdallas.edu)

Contents

[Introduction 2](#_Toc339131735)

[Enhanced Entity Relationship (EER) Diagram [UPDATED] 2](#_Toc339131736)

[EER Diagram to Relational Schemas 4](#_Toc339131737)

[Mapping Regular Entity Types, Specializations, and Unions 4](#_Toc339131738)

[Mapping Weak Entity Types 5](#_Toc339131739)

[Mapping Binary 1:1 Relationship Types 5](#_Toc339131740)

[Mapping Binary 1:N Relationship Types 5](#_Toc339131741)

[Mapping Binary N:M Relationship Types 6](#_Toc339131742)

[Mapping Multi-Valued Attributes 6](#_Toc339131743)

[Mapping of N-Ary Relationship Types 6](#_Toc339131744)

[Final Relational Schemas 6](#_Toc339131745)

[Documentation for Relational Schemas 7](#_Toc339131746)

[Conclusion 10](#_Toc339131747)

[Summary 10](#_Toc339131748)

[Future Work 10](#_Toc339131749)

# Introduction

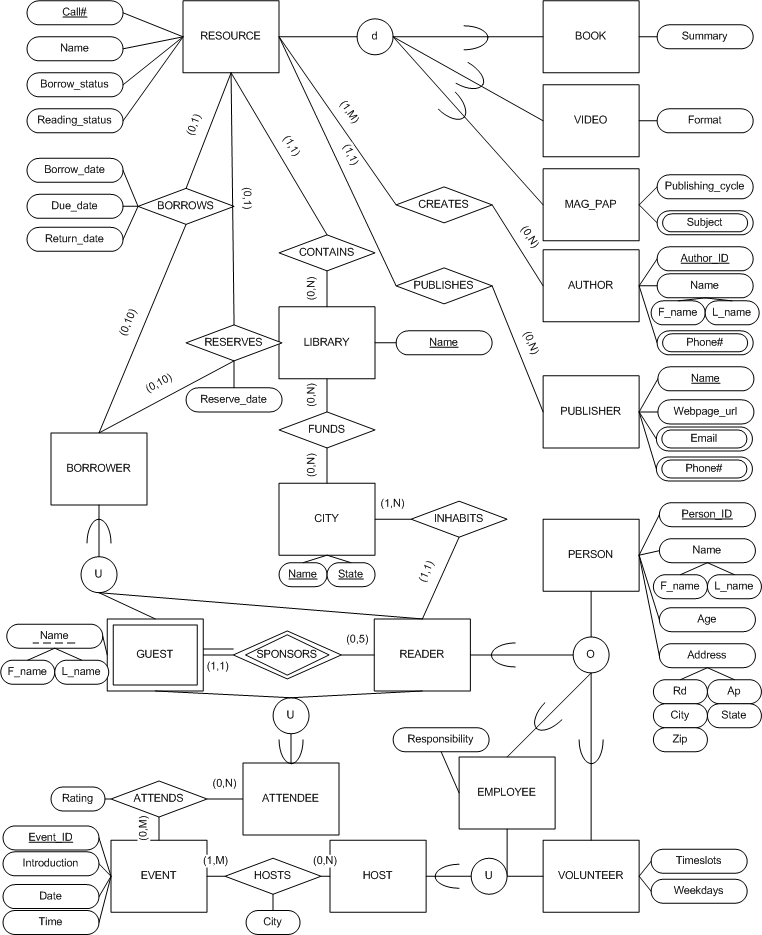
This report comprises four main sections:

1. EER Diagram UPDATED – In this section I show the updated EER diagram from phase 1. This has fixes from the phase 1 submission and from difficulties implementing the relational schema.
2. EER Diagram to Relational Schemas – In this section I step through the book’s seven-step algorithm for mapping an EER diagram to a relational schema.
3. Documentation for Relation Schemas – In this section I elaborate on the relational schemas and introduce data type constraints, data type formats, etc.
4. Conclusion – In this section I summarize the report. Assumptions, Limitations, and Explanations

# Enhanced Entity Relationship (EER) Diagram [UPDATED]

The updates are:

* Changed the line between SPONSORS and GUEST to a double line.



# EER Diagram to Relational Schemas

The text book uses a seven-step algorithm to completely map an EER diagram to a set of relational schemas. The following sections follow this algorithm step by step. The results of each step are accumulated and shown in the Final Relational Schemas section at the end. Referential integrity constraints are listed in the summary section.

## Mapping Regular Entity Types, Specializations, and Unions

In this section I map regular entity types, specializations, and union types. Any multi-valued attributes are left for a later step in the algorithm.

RESOURCE and its subclasses – each subclass gets its own relation

* RESOURCE( Call#, Name, Borrow\_status, Reading\_status )
* BOOK( R#, Summary )
* VIDEO( R#, Format )
* MAG\_PAP( R#, Publishing\_cycle )

AUTHOR( Author\_ID, F\_name, L\_name )

PUBLISHER( Name, Webpage\_url )

LIBRARY( Name )

CITY( Name, State )

PERSON and its subclasses – each subclass gets its own relation. The READER type has a local relationship and the EMPLOYEE and VOLUNTEER types are involved in a union relationship. It’s best to separate these into their own relations.

* PERSON( Person\_ID, F\_name, L\_name, Age, Rd, Ap, City, State, Zip )
* READER( P\_ID )
* EMPLOYEE( P\_ID, Responsibility )
* VOLUNTEER( P\_ID, Timeslots, Weekdays )

HOST is a union of the EMPLOYEE and VOLUNTEER types so it must have a surrogate key.

* HOST( Host\_ID, E\_ID, V\_ID )

ATTENDEE is a union of the GUEST and READER types so it must have a surrogate key.

* ATTENDEE( Attendee\_ID, G\_f, G\_l, R\_ID )

EVENT( Event\_ID, Introduction, Date, Time )

BORROWER is a union of the GUEST and READER types so it must have a surrogate key.

* BORROWER( Borrower\_ID, G\_f, G\_l, R\_ID )

## Mapping Weak Entity Types

The single weak entity type is a basic mapping. It uses a Person\_ID in conjunction with its F\_name and L\_name to uniquely identify GUESTs.

GUEST( F\_name, L\_name, R\_ID )

## Mapping Binary 1:1 Relationship Types

No binary 1:1 relationship types exist in the EER diagram.

## Mapping Binary 1:N Relationship Types

It usually is a good idea to merge 1:N relationship with the 1 side of the relationship to reduce the number of relations in the database.

PUBLISHES can be merged with the RESOURCE type since each RESOURCE may have only one PUBLISHER. RESOURCE updates to

* RESOURCE( Call#, Name, Borrow\_status, Reading\_status, P\_name )

CONTAINS can be merged with RESOURCE since each RESOURCE may be contained in only a single library at a time. RESOURCE updates to

* RESOURCE( Call#, Name, Borrow\_status, Reading\_status, P\_name, L\_name )

BORROWS can be merged with RESOURCE since each RESOURCE may be borrowed by only a single BORROWER at a time. However, since BORROWS has local attributes, it is better to make it another relation

* BORROWS( R#, B\_ID, Borrow\_date, Due\_date, Return\_date )

RESERVES can be merged with RESOURCE since each RESOURCE may be reserved by only a single BORROWER at a time. However since RESERVES has local attributes, it is better to make it another relation

* RESERVES( R#, B\_ID, Reserve\_date )

INHABITS can be merged with READER since each READER may only inhabit a single city at a time. READER updates to

* READER( P\_ID, C\_name, C\_state )

SPONSORS is the identifying relationship for a GUEST and READER. It can be merged with GUEST since each GUEST must be sponsored by only one READER. No updates are necessary to the GUEST type.

## Mapping Binary N:M Relationship Types

Each N:M relationship type gets its own relation.

CREATES( A\_ID, R# )

FUNDS( C\_name, C\_state, L\_name )

HOSTS( H\_ID, E\_ID, City )

ATTENDS( A\_ID, E\_ID, Rating )

## Mapping Multi-Valued Attributes

Each multi-valued attributes becomes its own relation.

MAG\_PAP\_SUBJECT( Mp#, Subject )

AUTHOR\_PHONE( A\_ID, Phone# )

PUBLISHER\_EMAIL( P\_name, Email )

PUBLISHER\_PHONE( P\_name, Phone# )

## Mapping of N-Ary Relationship Types

No non-binary relationship types exist in the EER diagram.

## Final Relational Schemas

|  |  |
| --- | --- |
| **RELATION** | **Referential (Integrity) Constraints** |
| RESOURCE(Call#, Name, Borrow\_status, Reading\_status, P\_name, L\_name) | P\_name -> PUBLISHER.Name  L\_name -> LIBRARY.Name |
| BOOK( R#, Summary ) | [FK] R# -> RESOURCE.Call# |
| VIDEO( R#, Format ) | [FK] R# -> RESOURCE.Call# |
| MAG\_PAP( R#, Publishing\_cycle ) | [FK] R# -> RESOURCE.Call# |
| AUTHOR( Author\_ID, F\_name, L\_name ) |  |
| PUBLISHER( Name, Webpage\_url ) |  |
| LIBRARY( Name ) |  |
| CITY( Name, State ) |  |
| PERSON( Person\_ID, F\_name, L\_name, Age, Rd, Ap, City, State, Zip ) |  |
| EMPLOYEE( P\_ID, Responsibility ) | [FK] P\_ID -> PERSON.Person\_ID |
| VOLUNTEER( P\_ID, Timeslots, Weekdays ) | [FK] P\_ID -> PERSON.Person\_ID |
| READER( P\_ID, C\_name, C\_state ) | [FK] P\_ID -> PERSON.Person\_ID  C\_name -> CITY.Name  C\_state -> CITY.State |
| GUEST( F\_name, L\_name, R\_ID ) | R\_ID -> READER.P\_ID |
| HOST( Host\_ID, E\_ID, V\_ID ) | [FK] E\_ID -> EMPLOYEE.P\_ID  [FK] V\_ID -> VOLUNTEER.P\_ID |
| ATTENDEE( Attendee\_ID, G\_f, G\_l, R\_ID ) | G\_f -> GUEST.F\_name  G\_l -> GUEST.L\_name  R\_ID -> READER.P\_ID |
| EVENT( Event\_ID, Introduction, Date, Time ) |  |
| BORROWER( Borrower\_ID, G\_f, G\_l, R\_ID ) | G\_f -> GUEST.F\_name  G\_l -> GUEST.L\_name  R\_ID -> READER.P\_ID |
| BORROWS( R#, B\_ID, Borrow\_date, Due\_date, Return\_date ) | [FK] R# -> RESOURCE.Call#  [FK] B\_ID -> BORROWER.Borrower\_ID |
| RESERVES( R#, B\_ID, Reserve\_date ) | [FK] R# -> RESOURCE.Call#  [FK] B\_ID -> BORROWER.Borrower\_ID |
| CREATES( A\_ID, R# ) | [FK] A\_ID -> AUTHOR.Author\_ID  [FK] R# -> RESOURCE.Call# |
| FUNDS( C\_name, C\_state, L\_name ) | [FK] C\_name -> CITY.Name  [FK] C\_state -> CITY.State  [FK] L\_name -> LIBRARY.Name  [FK] {C\_name, C\_state} -> {CITY.Name, CITY.State} |
| HOSTS( H\_ID, E\_ID, City ) | [FK] H\_ID -> HOST.Host\_ID  [FK] E\_ID -> EVENT.Event\_ID |
| ATTENDS( A\_ID, E\_ID, Rating ) | [FK] A\_ID -> ATTENDEE.Attendee\_ID  [FK] E\_ID -> EVENT.Event\_ID |
| MAG\_PAP\_SUBJECT( Mp#, Subject ) | [FK] Mp# -> MAG\_PAP.R# |
| AUTHOR\_PHONE( A\_ID, Phone# ) | A\_ID -> AUTHOR.Author\_ID |
| PUBLISHER\_EMAIL( P\_name, Email ) | P\_name -> PUBLISHER.Name |
| PUBLISHER\_PHONE( P\_name, Phone# ) | P\_name -> PUBLISHER.Name |

# Documentation for Relational Schemas

In this section I elaborate on the relational schemas created in the previous section. I introduce data types and constraints for attributes of relations. There are some database-wide rules for data types as follows:

* Dates are all in form “MM-DD-YYYY” as a string of characters.
* Times are all in form “HH:MM:SS” as a string of characters.
* Phone numbers are all in “xxx-xxx-xxxx” as a string of characters.
* IDs are numbers generated using the auto increment feature of a database system.
* Person\_ID is generated by using the first letter of the F\_name, a random character, the first letter of the L\_name, and a 6 digit random integer.

|  |  |  |
| --- | --- | --- |
| **RELANTION** | **Attributes** | **Data type and constraints** |
| RESOURCE | Call# | string, 8 chars, non-null, unique |
| Name | string <= 60 chars |
| Borrow\_status | string <= 20 chars; “available” or “unavailable” |
| Reading\_status | string <= 20 chars; “for borrow” or “in library reading only” |
| P\_name | string <= 60 chars |
| L\_name | string <= 60 chars |
| BOOK | R# | string, 8 chars, non-null, unique |
| Summary | text <= 500 chars |
| VIDEO | R# | string, 8 chars, non-null, unique |
| Format | string <= 10 chars; “VCD”, “DVD”, “cassette”, “USB” |
| MAG\_PAP | R# | string, 8 chars, non-null, unique |
| Publishing\_cycle | string <= 20 chars; “bi-weekly”, “monthly”, “bi-annually”, “annually” |
| AUTHOR | Author\_ID | string, 5 chars; [“00001”, “99999”], non-null, unique |
| F\_name | string <= 20 chars |
| L\_name | string <= 20 chars |
| PUBLISHER | Name | string <= 30 chars, non-null, unique |
| Webpage\_url | string <= 60 chars |
| LIBRARY | Name | string <= 60 chars, non-null, unique |
| CITY | Name | string <= 60 chars, non-null |
| State | string <= 15 chars, non-null |
| {Name, State} | unique |
| PERSON | Person\_ID | string, 9 chars; {f + c + l + xxxxxx}, non-null, unique |
| F\_name | string <= 60 chars, non-null |
| L\_name | string <= 60 chars, non-null |
| Age | integer |
| Rd | string <= 30 chars |
| Ap | string <= 30 chars |
| City | string <= 60 chars |
| State | string <= 15 chars |
| Zip | integer |
| EMPLOYEE | P\_ID | string, 9 chars; {f + c + l + xxxxxx}, non-null, unique |
| Responsibility | string <= 60 chars |
| VOLUNTEER | P\_ID | string, 9 chars; {f + c + l + xxxxxx}, non-null, unique |
| Timeslots | integer |
| Weekdays | integer |
| (Age) | integer <= 75 |
| READER | P\_ID | string, 9 chars; {f + c + l + xxxxxx}, non-null, unique |
| C\_name | string <= 60 chars, non-null |
| C\_state | string <= 15 chars, non-null |
| GUEST | F\_name | string <= 60 chars, non-null |
| L\_name | string <= 60 chars, non-null |
| R\_ID | string, 9 chars; {f + c + l + xxxxxx}, non-null |
| {F\_name, L\_name, R\_ID} | unique |
| HOST | Host\_ID | integer, non-null, unique, auto-increment |
| E\_ID | string, 9 chars; {f + c + l + xxxxxx} |
| V\_ID | string, 9 chars; {f + c + l + xxxxxx} |
| {E\_ID, V\_ID} | at least one is non-null |
| ATTENDEE | Attendee\_ID | integer, non-null, unique, auto-increment |
| G\_f | string <= 60 chars |
| G\_l | string <= 60 chars |
| R\_ID | string, 9 chars; {f + c + l + xxxxxx} |
| {{G\_f, G\_l}, R\_ID} | at least one is non-null |
| EVENT | Event\_ID | integer, non-null, unique, auto-increment |
| Introduction | text <= 500 chars |
| Date | string, 10 chars, “MM/DD/YYYY” |
| Time | string, 8 chars, “HH:MM:SS” |
| BORROWER | Borrower\_ID | integer, non-null, unique, auto-increment |
| G\_f | string <= 60 chars |
| G\_l | string <= 60 chars |
| R\_ID | string, 9 chars; {f + c + l + xxxxxx} |
| {{G\_f, G\_l}, R\_ID} | at least one is non-null |
| BORROWS | R# | string, 8 chars, non-null |
| B\_ID | integer, non-null |
| Borrow\_date | string, 10 chars, “MM/DD/YYYY” |
| Due\_date | string, 10 chars, “MM/DD/YYYY”, >= Borrow\_date |
| Return\_date | string, 10 chars, “MM/DD/YYYY”, >= Borrow\_date |
| {R#, B\_ID} | unique |
| RESERVES | R# | string, 8 chars, non-null |
| B\_ID | integer, non-null |
| Reserve\_date | string, 10 chars, “MM/DD/YYYY” |
| {R#, B\_ID} | unique |
| CREATES | A\_ID | string, 5 chars; [“00001”, “99999”], non-null |
| R# | string, 8 chars, non-null |
| {A\_ID, R#} | unique |
| FUNDS | C\_name | string <= 60 chars, non-null |
| C\_state | string <= 15 chars, non-null |
| L\_name | string <= 60 chars, non-null |
| {C\_name, C\_state, L\_name} | unique |
| HOSTS | H\_ID | integer, non-null |
| E\_ID | string, 9 chars; {f + c + l + xxxxxx}, non-null |
| City | string <= 60 chars |
| {H\_ID, E\_ID} | unique |
| ATTENDS | A\_ID | integer, non-null |
| E\_ID | integer, non-null |
| Rating | integer [0,10] |
| {A\_ID, E\_ID} | unique |
| MAG\_PAP\_SUBJECT | Mp# | string, 8 chars, non-null |
| Subject | string <= 20 chars, non-null |
| {Mp#, Subject} | unique |
| AUTHOR\_PHONE | A\_ID | string, 5 chars; [“00001”, “99999”], non-null |
| Phone# | string, 12 chars; “xxx-xxx-xxxx”, non-null |
| {A\_ID, Phone#} | unique |
| PUBLISHER\_EMAIL | P\_name | string <= 30 chars, non-null |
| Email | string <= 40 chars, non-null |
| {P\_name, Email} | unique |
| PUBLISHER\_PHONE | P\_name | string <= 30 chars, non-null |
| Phone# | string, 12 chars; “xxx-xxx-xxxx”, non-null |
| {P\_name, Phone#} | unique |

# Conclusion

## Summary

In this report I updated the EER diagram from phase 1 and mapped it to relational schemas using the book’s seven-step algorithm. I also elaborated on the data types for the relational schemas.

## Future Work

This report derived the relational model of the ABC Library database system from the EER diagram. The next step is to create SQL queries to create the database in a real system. For further questions contact me at the email address on the title page.